Argonne's BlueGene/P Supercomputer

Software Overview

Vitali Morozov, Ray Loy, and Kalyan Kumaran

Application Performance and Data Analytics

Argonne Leadership Computing Facility

Argonne NATIONAL LABORATORY

... for a brighter future

U.S. Department of Energy

UChicago Argonne LLC

Office of Science U.S. DEPARTMENT OF ENERGY

DOE Leadership Computing Facility Strategy

- DOE SC selected the ORNL, ANL and PNNL team (May 12, 2004) based on a competitive peer review of 4 LCF proposals
  - ORNL will deploy a series of systems based on Cray's XT3/4 architectures @ 250TF/s in FY07 and 1000TF/s in FY08/9
  - ANL will develop a series of systems based on IBM's BlueGene @ 100TF/s in FY07 and 250-500TF/s in FY08/FY09 with IBM Blue Gene/P
  - PNNL will contribute software technology
- DOE SC will make these systems available as capability platforms to the broad national community via competitive awards (e.g. INCITE Allocations)
  - Each facility will target ~20 large-scale production applications teams
  - Each facility will also support development users
- DOE's LCFs complement existing and planned production resources at NERSC
  - Capability runs will be migrated to the LCFs, improving NERSC throughput
  - NERSC will play an important role in training and new user identification

Mission and Vision for the ALCF

Our Mission

Provide the computational science community with a world leading computing capability dedicated to breakthrough science and engineering.

Our Vision

A world center for computation driven scientific discovery that has:

- outstandingly talented people,
- the best collaborations with computer science and applied mathematics,
- the most capable and interesting computers and,
- a true spirit of adventure.

See http://www.alcf.anl.gov/ for info and openings

ALCF Timeline

2004

- Formed of the Blue Gene Consortium with IBM
- DOE-SC selected the ORNL, ANL and PNNL team for Leadership Computing Facility award

2005

- Installed 5 teraflops Blue Gene/L for evaluation

2006

- Began production support of 6 INCITE projects, with BGW
- Continued code development and evaluation
- “Lehman” Peer Review of ALCF campaign plans

2007

- Increased to 9 INCITE projects; continued development projects
- Installed 100 teraflops BlueGene/P (late 2007)

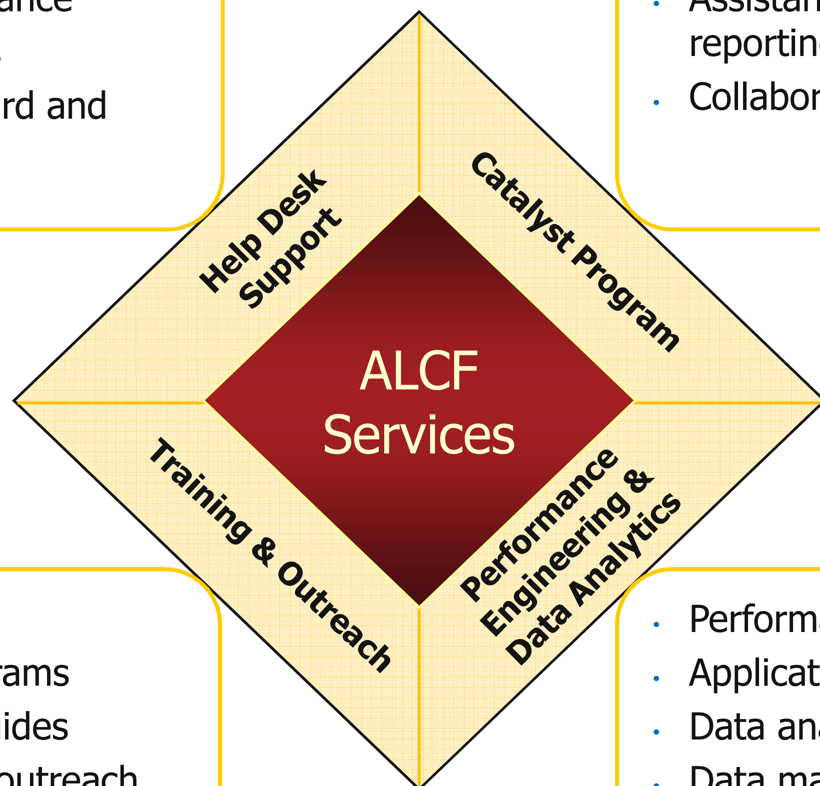
2008

- Began support of 20 INCITE projects on BG/P
- Added 450 teraflops BG/P

ALCF Service Offerings

- Startup assistance
- User administration assistance
- Job management services
- Technical support (Standard and Emergency)

- ALCF science liaison
- Assistance with proposals, planning, reporting
- Collaboration within science domains



- Workshops & seminars
- Customized training programs
- On-line content & user guides
- Educational and industry outreach programs

- Performance engineering
- Application tuning
- Data analytics
- Data management services

Overview

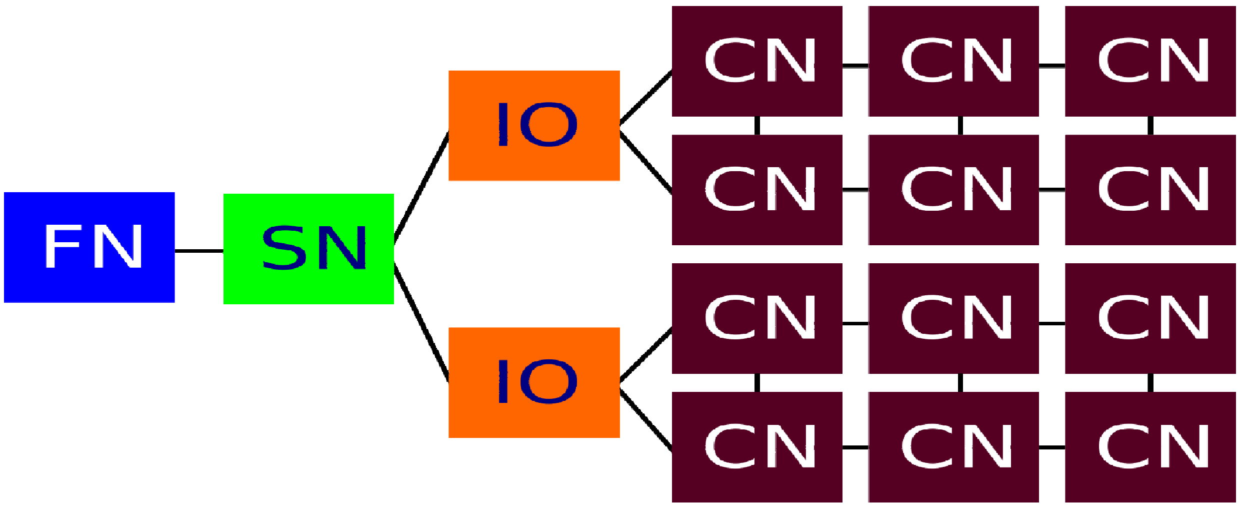
- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

Configuration Details

- Login Servers
  - compile and submit jobs to ANL's Cobalt scheduler
  - surveyor.alcf.anl.gov – 13.9T 1-rack BG/P system - testing and development, in production mode
  - intrepid.alcf.anl.gov – 8-rack BG/P production system - open for all INCITE users
  - intrepid.alcf.anl.gov – 32-rack BG/P system - open for Early Science applications
- Service Nodes
  - users have restricted access
  - jobs are started from here
  - executable and working directory must be accessible
- I/O Nodes
  - 1/64 IO nodes / compute nodes ratio
  - each compute node is mapped to particular IO node
- Compute Nodes [1024 nodes per rack]
  - users have no access
- Storage Services
  - users have no access

BlueGene/P Software Organization

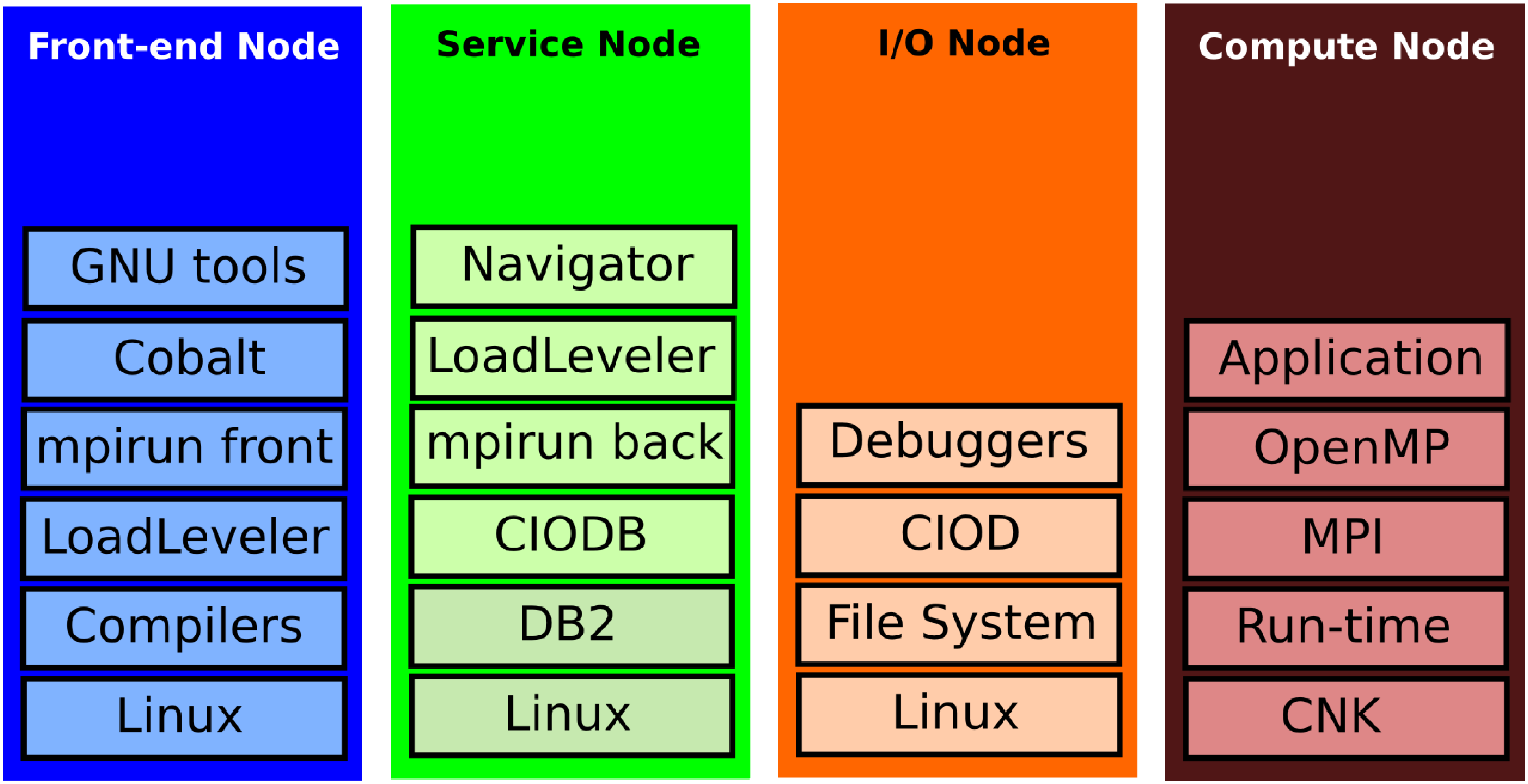
- **Front-end nodes (FN)**, dedicated for user's to login, compile programs, submit jobs, query job status, debug applications
- **Service nodes (SN)**, perform system management services, create and monitoring processes, initialize and monitor hardware, configure partitions, control jobs, store statistics
- **I/O nodes (IO)**, provide a number of OS services, such as files, sockets, process management, debugging
- **Compute nodes (CN)**, run user application, limited OS services



BlueGene/P Programming Environment

- **Linux cross-compilation environment**
  - users login to FEN for compilation, job submission, debugging
- **Space sharing**
  - exactly one job per partition
  - smp-mode, one MPI task/node, 4 threads/task, 2GB of RAM
  - dual-mode, two MPI tasks/node, 2 threads/task, 1GB of RAM
  - vn-mode, 4 single-threaded MPI tasks/node, 512MB of RAM
- **Fortran, C, C++ compilers, MPI, OpenMP**
  - memory limited to physical memory
  - statically and dynamically linked libraries
  - restricted set of POSIX routines (no fork, system, …)
  - threading support
  - MPI based on ANL's mpich2
- **SPMD model**
  - compute nodes run the same executable

BlueGene/P Software Stack



Application Developer's view

- **4 CPU core per node, 850 MHz, each core can do up to two double multiply/add instructions per cycle**
  - peak performance is 3.4 GFlops/core, 13.6 GFlops/node
- **3D torus network**
  - point to point MPI_SEND, MPI_RECV
  - deterministic protocol for short messages
  - deterministic eager protocol for medium messages
  - adaptive rendezvouz protocol for long messages
- **Global tree network**
  - efficient implementation of all-to-one, one-to-all, and all-to-all calls
- **Global interrupt network**
  - fast MPI_BARRIER

Overview

- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

Building Executable: MPI-Wrapper

- MPI wrappers to IBM compiler set

mpixlc	mpixlcxx	mpixlf77	mpixlf90	mpixlf2003

- Thread-safe versions of MPI wrappers to IBM compiler set

mpixlc_r	mpixlcxx_r	mpixlf77_r	mpixlf90_r	mpixlf2003_r

- MPI wrappers to GNU compiler set

mpicc	mpicxx	mpif77

- BlueGene/L users: change your scripts

mpicc.ibm -> mpixlc	mpicxx.ibm -> mpicxx	mpif77.ibm -> mpixlf77
mpicc.gnu -> mpicc	mpicxx.gnu -> mpicxx	mpif77.gnu -> mpif77

Sample BlueGene/P makefile

```
BGPDRIVER = /bgsys/drivers/ppcfloor
CC = $(BGPDRIVER)/comm/bin/mpixlc
CXX = $(BGPDRIVER)/comm/bin/mpixlcxx
FC = $(BGPDRIVER)/comm/bin/mpixlf90
OPTFLAGS = -O3 -qarch=450d -qtune=450 -qhot
CFLAGS = -qlist -qsource -qreport -g
FFLAGS = -qlist -qsource -qreport -g

myprog:     myprog.o
      $(FC) $(FFLAGS) -o myprog myprog.o
```

Building Executable: Direct Compiler

/usr/bin/bgcc -> /opt/ibmcmp/vacpp/bg/9.0/bin/bgcc

bgxlc, bgxlc_r	compile C source file
bgxlc++, bgxlc++_r, bgxlC, bgxlC_r	compile C++ source file
bgcc, bgcc_r	compile pre-ANSI C non-standard source file
bgc89, bgc89_r	compile C89-conformed C source file
bgc99, bgc99_r	compile C99-conformed C source file
bgxlf, bgxlf_r, bgf77, bgfort77	compile Fortran 77 source file
bgxlf90, bgxlf90_r, bgf90	compile Fortran 90 source file
bgxlf95, bgxlf95_r, bgf95	compile Fortran 95 source file
bgxlf2003, bgxlf2003_r, bgf2003	compile Fortran 2003 source file

```
DRIVER_PATH=/bgsys/drivers/ppcfloor
bgxlC -o MPI_Prog MPI_Prog.C  -I$DRIVER_PATH/comm/include/ \
   -L$DRIVER_PATH/comm/lib/ -lcxxmpich.cnk -lmpich.cnk -ldcmfcoll.cnk \
    -ldcmf.cnk -lpthread -lrt  -L$DRIVER_PATH/runtime/SPI -lSPI.cna
```

OpenMP Implementation

- Shared-memory parallelism is supported on single node
- Interoperability with MPI as
  - MPI at outer level, across compute nodes
  - OpenMP at inner level, within a compute node
- Thread-safe compiler version should be used
  - with any threaded/OMP/SMP applications
- OpenMP 2.5 standard directives are supported:
  - parallel, for, parallel for, sections, parallel sections, critical, single
  - #pragma omp <rest of pragma> for C/C++
  - !$OMP <rest of directive> for Fortran
- Compiler functions
  - omp_get_num_procs, omp_get_num_threads
    omp_get_thread_num, omp_set_num_threads

Overview

- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

Common Approaches to Application I/O

- Single process - root - performs I/O
  - trivially simple to implement
  - limited bandwidth equal to one client's performance
  - insufficient memory and delays in root to keep data
- All processes write to its own file
  - no synchronization between tasks
  - avoids file system sharing
  - very many files may be created
  - difficulty to post-process data
  - bottlenecks from I/O hardware
- All processes access single file
  - a single file to manage
  - post-processing can be avoided
  - possible file system sharing and other inefficiencies
  - bottlenecks from I/O hardware

Parallel I/O in HPC

- Applications want to achieve scalability, parallelism, high bandwidth, and usability
- Applications require more software than just a parallel file system
- Multiple layers are provided with distinct roles:
  - Parallel file system
    - *maintains logical space, provides efficient access to data (PVFS, GPFS)*
  - I/O forwarding
    - *assists with I/O scaling issues, load balance for I/O servers*
  - Middleware
    - *organizes access by many processes (MPI-IO)*
  - High-level I/O library
    - *maps application abstractions to a structured portable data format (HDF5, Parallel netCDF)*

I/O on BlueGene/P

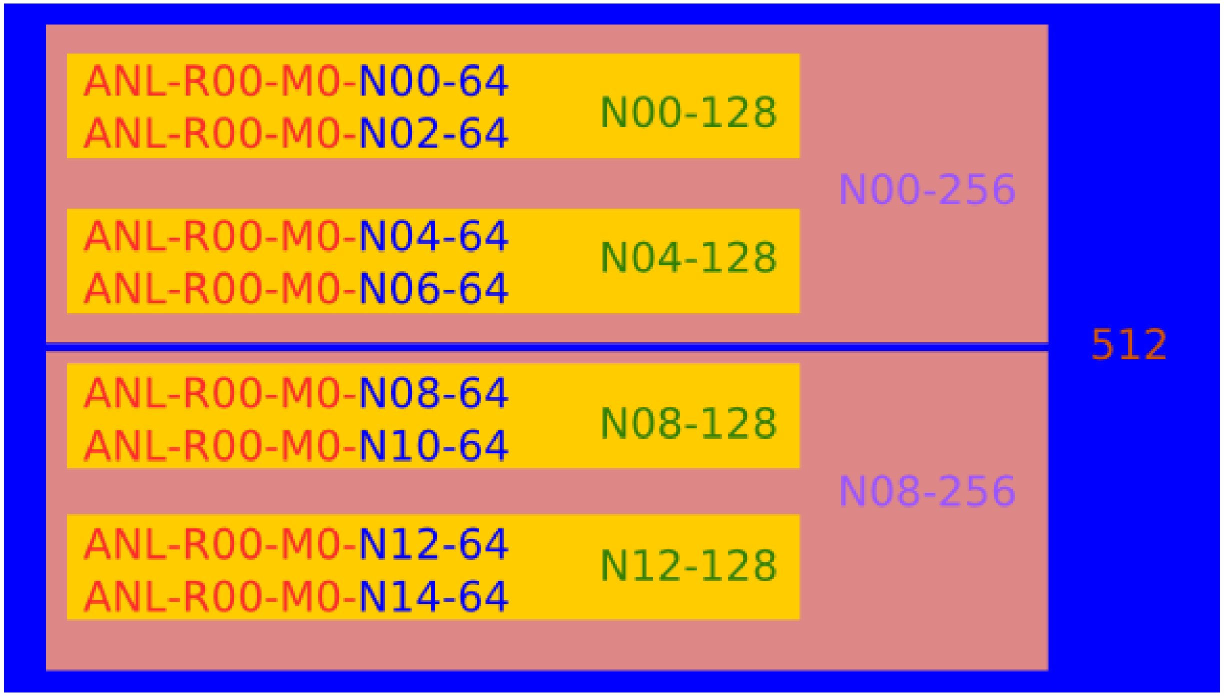
- Home directory
  - GPFS
  - /gpfs/home/<username> -> /home/<username>
  - extra space in /gpfs1 if needed
  - visible from login, compute, I/O, and service nodes
  - limited in space
  - daily snapshots in ~/.snapshots

- Data
  - PVFS
  - /pvfs-surveyor
  - visible from login, I/O, and compute nodes
  - invisible from the service nodes, so, cannot contain exec files
  - scratch data space, no backups

Overview

- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

BlueGene/P Partitions



- Minimal partition size is 64 nodes: due to one I/O 64 compute node ratio
- Larger partitions are configured by combining smaller ones
- If a job is running on a partition, no other job can run on the enclosing larger partitions
- Not all partitions are available at all times
- bg-listblocks --all lists all defined partitions

Cobalt: An ANL's Scheduler for HPC*

- Research in nature - investigating advanced systems management for complex cutting-edge architectures
- Open source and uses open source components enabling rapid experimentation and exploration advanced features
- Focuses on reconfigurable environments for user and growing hardware
- Fits to both computational needs and computer science research (most resource managers are not system software research environments)
- Smaller and simpler is better (4K lines of Python code, dynamic kernel selection, different I/O node kernels, different kernel tuning parameters, flexibility and configurability, small partition support)

* http://www-unix.mcs.anl.gov/cobalt/index.xml

Resource Manager and Job Scheduler

- Cobalt supports standard commands to manage jobs

qsub: submit a job	qstat: query a job status
qdel: delete a job	qalter: alter batched job parameters

- Different queues
  - short: 24x7x365, <60 minute jobs, higher priority weekdays 8am-2pm CST
  - medium: 24x7x365, <3 hour jobs, any size, higher priority weekdays 2pm-8pm CST
  - long: <6 hour jobs, 8pm-8am weekdays and full day weekends
  - develop: development jobs of 512 nodes or less
- FIFO based scheduler
  - chooses the best fit from the top of the queue
- Maintenance Day: Monday
- Reservations used for special needs

qsub: Submitting a Job

Type qsub

Usage: qsub [-d] [-v] -A <project name> -q <queue> --cwd <working directory>
--env envvar1=value1:envvar2=value2 --kernel <kernel profile>
-K <kernel options> -O <outputprefix> -t time <in minutes>
-e <error file path> -o <output file path> -i <input file path>
-n <number of nodes> -h --proccount <processor count>
--mode <mode> <command> <args>

-t <time_in_minutes>	required runtime
-n <number_on_nodes>	number of nodes
--proccount <number_of_cores>	number of CPUs
--mode <smp|dual|vn>	running mode
--env VAR1=1:VAR2=1	environment variables
<command> <args>	command with arguments

Do not give a partition: it is chosen by a scheduler
If fit to a sooner-to-schedule, a queue is adjusted automatically

qsub: Examples of Submitting a Job

- Despite being redundant, we recommend to always specify the number of nodes, the number of CPUs, and the mode of your run

- qsub -q short -t 10 -n 64 --proccount 64 --mode smp Hello
  - submits a job to a short queue
  - will run no longer than 10 minutes or when executable stops
  - will use smp-mode with 64 nodes, 64 CPUs

- qsub -q short -t 10 -n 4 --proccount 16 --mode vn -O My_Run My_Exe My_File
  - submits a job to a short queue and run no longer than 10 minutes
  - will use vn-mode with 4 nodes, 16 CPUs
  - will allocate 64-node partition, 60 nodes will stay unused
  - will run program My_Exe with argument My_File
  - will create My_Run.output as stdout and My_Run.error as stderr files

qsub: A Script to Submit a Typical Job

```
#!/bin/bash

RUN=<program_executable>
NODES=64
CORES=256
MODE=vn
MAPPING=XYZT

TASK=$RUN-$NODES-$CORES-$MODE

rm -rf $TASK.error $TASK.output

echo Processors: nodes $NODES, cores $CORES, mode $MODE

qsub -q short -t 0:10:00 -n $NODES --proccount $CORES --mode $MODE -O $TASK \
   --env BG_MAPPING=$MAPPING $RUN

qstat -f
touch $TASK.error
tail -f $TASK.error
```

qstat: Show Status of a Batch Job(s)

- qstat -f <job_id1> <job_id2>
  - a full display is produced

```
JobID    JobName     User    WallTime QueuedTime RunTime Nodes State   Location     Mode Procs Queue StartTime
==========================================================================================================
11543 fl-64-64-smp morozov 00:30:00 00:00:06 00:13:41 64 running ANL-R00-M1-N02-64 SMP 64 short 02/27/08
```

  - job_id can be used to kill the job of alter the job parameters
  - valid status: queued, running
  - check the mode of your job

- qstat -Q
  - will show all available queues and their limits
  - special queues, which we use to handle reservations

qdel: Kill a Job

- qdel <jobid1> <jobid2>
  - delete the job from a queue
  - terminated a running job

qalter, qmove: Alter Parameters of a Job

- Allows to alter the parameters of both queued and running jobs
- Very useful for the running jobs, which would unexpectedly coming to exceed their allocated time

- Type qalter

Usage: qalter [-d] [-v] -A <project name> -t <time in minutes>

-e <error file path> -o <output file path>

-n <number of nodes> -h --proccount <processor count>

-M <email address> --mode <mode smp/dual/vn> <jobid1> <jobid2>

- Careful: -t <time in minutes>:
  – it is NOT the time left for the running jobs!
  – it is elapsed time since the beginning of the run, after which Cobalt kills the job
- use qmove to change the queue

Why a job is not running in a queue

- there is a reservation, which interferes with your job
  - showres shows all reservations currently in place
- there is no available partitions
  - partlist shows all partitions marked as functional
  - partlist shows the assignment of each partition to a queue
- wrong queue
  - the job submitted to a queue, which is restricted to run at this time
- partitions are not freed
  - in specific situations, a job quits and does not free a partition => a partition is treated as busy, but there is no job, which holds this partition
  - bg-listblocks --all --long prints full information of all blocks
  - the state is identified by a combination of qstat -f, bg-listblocks

Overview

- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

Tools: Improved Performance, Profiling, Debugging …

- Most tools are under /soft/apps
- Improved performance with optimized libraries
  - BLAS/LAPACK versus LibGOTO/LAPACK
  - BlueGene optimized Mass, MassV, ESSL libraries from IBM
- Practical Optimization
  - compiler switches
  - profiling and profiling tools: HPCT, Profiling “-pg”, “-qdebug=function_trace”, TAU
- Tracing MPI_Barrier/printf/exit/abort standard debugging methods
- GDB / Totalview
  - the last choice, requires advanced experience

Optimization Steps w/o Code Changes

- Start from original MPI program, make it run
  - The least aggressive compiler options
  - Default libraries
- Increase compiler optimization options
- Verify different running modes: smp vs. dual vs. vn
- Use highly optimized libraries (BLAS-LibGOTO, MASSV, ESSL)
- Optimize communication performance: DCMF_EAGER
- Optimize mapping (logical MPI-task to CPU allocation): BG_MAPPING

Optimization Steps with Code Changes

- Use compiler directives
  - Alignment, aliasing, loop unrolling, SIMD vectorization
- Profiling (identify the bottleneck)
  - Profiling Tools with and without code modification
  - Use of hardware counters
  - Start code changes only if the bottleneck is concentrated
- Rearranging memory hierarchy
  - Ordered memory inquires improve cache reuse (Fortran N-dim arrays)
  - Use of contiguous memory blocks allows quadword loads
- Use double-hummer instructions
  - Available for Fortran, C, C++ as regular calls
  - Register/instructions scheduler is done by compiler
- Last choice: hand-coding assembly
  - Assembly generated by a compiler is a great help to understand the code

Memory Hierarchy

- L1 Instruction and L1 Data caches
  - 32 KB total size, 32-Byte line size, 64-way associative, round-robin
  - -qcache=level=1:type=d:assoc=64:line=32:size=32:\

    level=1:type=i=assoc=64:line=32:size=32
- L2 Data cache
  - 2KB prefetch buffer, 16 lines, 128-byte a line
  - -qcache=level=2:type=c:line=128:size=2
- L3 Data cache
  - 8 MB, 50 cycles latency
  - -qcache=level=3:type=c:line=128:size=8192:cost=50
- Memory size
  - 2GB DDR-2 at 425 MHz, 100 cycles
- Memory bandwidth
  - in L1-cache: ffpdx/stfpdx instructions, 1 quadword load/cycle: 16B*850 /s = 13.6 GB/s
  - out of L1-cache: complex memory hierarchy

IBM XL Compiler General Optimization

- Default: -qarch=[450|450d] -qnoautoconfig -qstaticlink -qtune=450
- -O0: no optimization, implies -qstrict_induction (no loop counter optim)
- -O = -O2: balanced optimization, implies -qstrict_induction -qstrict
- -O3 -qstrict: preserves program semantics
- -O3 = -O2 -qfloat=fltint:rsqrt:norngchk -qmaxmem=1 -qhot=level=1: aggressive but reasonably stable level
- -qhot: turns on High-Order loop analysis and Transformation unit
  - arraypad, level, simd, vector
- -qreport: produces a listing, shows how code was optimized
- -qipa: interprocedural analysis, use with caution
  - level, inline, list

IBM XL Compiler BG-Specific Optimization

- Architecture flags
  - -qalign: Fortran only, specifies the alignment of data
  - -qarch=450: generates PPC450 instructions
  - -qarch=450d: generates double-hummer instructions
- Increase of optimization aggressiveness
  - -O0 -qarch=450d: default optimization level
  - -O3 -qarch=450/450d
  - -O4 -qarch=450d -qtune=450
  - -O4 = -O3 -qarch -qtune -qcache -qhot -qipa=level=1
  - -O5 = -O4 -qipa=level=2
- -qlistopt: generates the listing with all flags used in compilation

Example program

```
#define SIZE 1024

double A[SIZE][SIZE];
double B[SIZE][SIZE];
double C[SIZE][SIZE];

double multiply(void)
{
  int i, j, k;

  for (i = 0; i < SIZE; i ++)
    for (j = 0; j < SIZE; j++)
      for (k = 0; k < SIZE; k++)
       C[i][j]
         += A[i][k] * B[k][j];

  return C[SIZE-10][SIZE-10];
}
```

-qreport: shows, how sections of code have been optimized

```
do {
   /* id=3 guarded */ /* ~10 */
   /* region = 52 */
   /* bump-normalized */
   /* independent */
   $.CSE15 = $.ICM0 + $.CIV3;
   $.CSE17 = B[$.ICM3][$.CSE15];
   $.CSE16 = C[$.ICM6][$.CSE15] + $.ICM7 * $.CSE17;
   C[$.ICM6][$.CSE15] = $.CSE16;
   $.CSE18 = B[$.ICM8][$.CSE15];
   C[$.ICM6][$.CSE15] = $.CSE16 + $.ICM9 * $.CSE18;
   $.CSE19 = C[$.ICMA][$.CSE15] + $.ICMB * $.CSE17;
   C[$.ICMA][$.CSE15] = $.CSE19;
   C[$.ICMA][$.CSE15] = $.CSE19 + $.ICMC * $.CSE18;
   $.CIV3 = $.CIV3 + 1;
} while ((unsigned) $.CIV3 < (unsigned) $.ICME);
```

Example program

- -qsource: produces a listing with source section
- -qlist: produces an object listing

```
lfpdx    fp4,fp36=B[]0(gr21,gr29,0,offset=8)
addi     gr21=gr21,32
lfpdx    fp3,fp35=B[]0(gr21,gr24,0,offset=-8)
fxcpmadd fp1,fp33=fp7,fp39,fp0,fp32,fp10,fp10,fcr
fxcpmadd fp6,fp38=fp9,fp41,fp0,fp32,fp11,fp11,fcr
fxcpmadd fp0,fp32=fp5,fp37,fp4,fp36,fp12,fp12,fcr
fxcpmadd fp4,fp36=fp2,fp34,fp4,fp36,fp13,fp13,fcr
fxcpmadd fp2,fp34=fp1,fp33,fp3,fp35,fp12,fp12,fcr
fxcpmadd fp1,fp33=fp6,fp38,fp3,fp35,fp13,fp13,fcr
stfpdx    C[]0(gr22,gr30,0,offset=-8184)=fp0,fp32
stfpdx    C[]0(gr22,gr29,0,offset=8)=fp4,fp36
```

Runtime Mode

- SMP mode
  - qsub --mode smp
  - Single MPI task on CPU0 / 2 GB RAM
- Dual mode
  - qsub --mode dual
  - Two MPI tasks on a node / 1GB RAM each
- Virtual Node mode
  - qsub --mode vn
  - Four MPI tasks on a node / 512 MB RAM each

Threading Support

- OpenMP is supported
  - NPTL pthreads implementation in glibc requires NO modifications

- Compute Note Kernel supports
  - execution of one quad-threaded process

    (each of the CPUs is assigned to each of maximum 4 threads)
  - execution of two two-threaded processes
  - execution of four single-threaded processes
  - proper mode should be specified for qsub

MPI Mapping

- Default XYZT mapping
  - (XYZ) are torus coordinates, T is a CPU number
  - X-coordinate is increasing first, then Y, then Z
  - All XYZT permutations are possible
- qsub --env BG_MAPPING=TXYZ --mode vn …
  - This puts MPI task 0,1,2,3 to Node 0 CPU0, CPU1, CPU2, CPU3; MPI tasks 4,5,6, and 7 to Node2 CPU0,CPU1,CPU2,CPU3
  - Typically, default XYZT is less efficient than TXYZ mapping
- qsub --BG_MAPPING=<FileName> --mode smp …
  - use high-performance toolkits to determine communication pattern
  - optimize mapping by custom mapfile
  - mapfile: each line contains 4 coordinates to place the task, first line for task 0, second line for task 1…
  - avoid conflict in mapfiles (no verification)

Optimized libraries

- BG-optimized BLAS Level 1,2,3 library from Kazushige Goto, U. of Texas
- IBM ESSL library: BLAS1, 2, 3 in /soft/apps/ESSL
- Generic versions of BLAS/LAPACK/FFTW

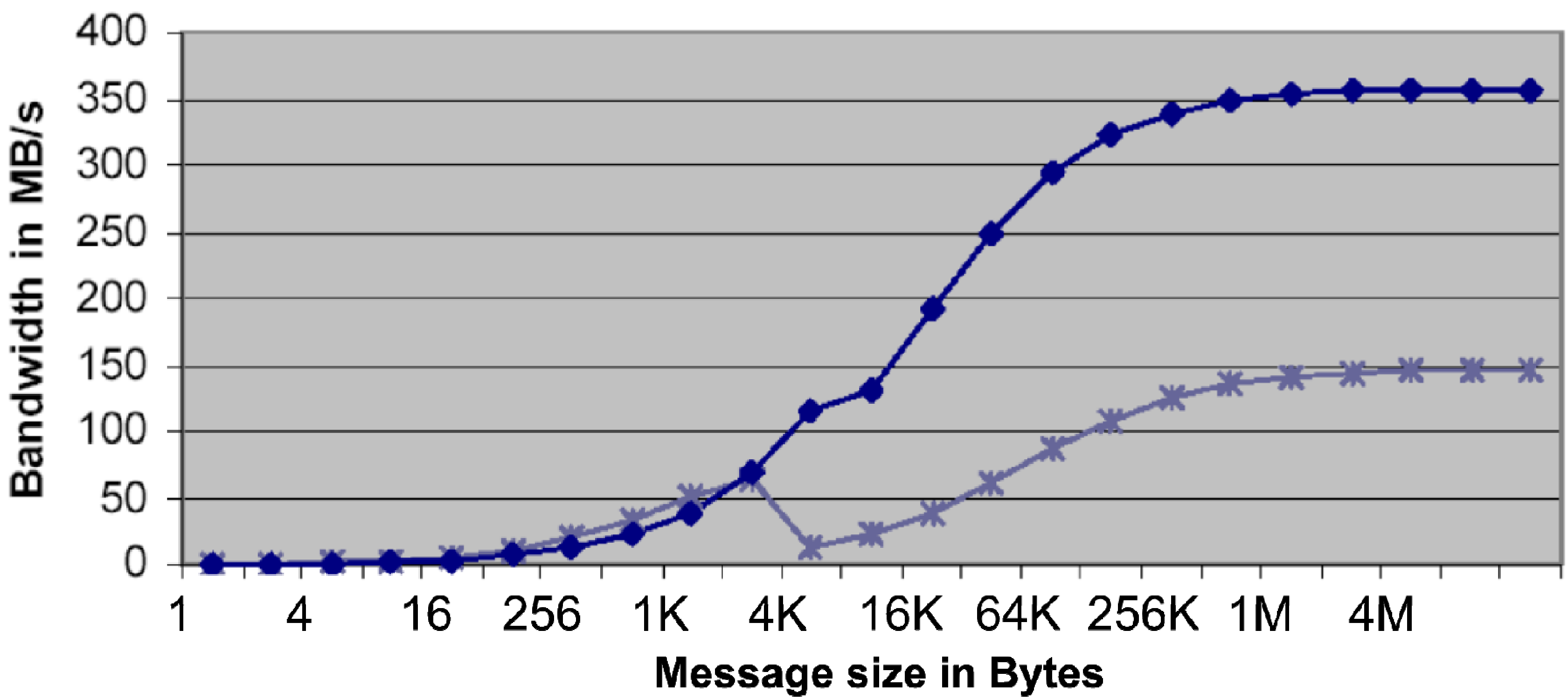
-O0	102.82 s	20.88 MFlop/s	
-O2	70.86 s	30.31 MFlop/s	
-O3	3.471 s	618.52 MFlop/s	
-O4	7.921 s	271.10 MFlop/s	
-O5	7.919 s	271.12 MFlop/s	
ESSL	0.836 s	2569.6 MFlop/s	75.76 % of peak
GOTO	0.828 s	2593.0 MFlop/s	76.26 % of peak

Communication Operations

- Best if no use of complex derived data
- For performance reason, it is advisable do not overlap p2p and collective operations
- P2P operations a figure from Application Development
  - Routing messages statically or dynamically
  - Control routing by DCMF_EAGER variable (changes the rendezvous threshold)
- Collective operations: latency and bandwidth from Application Development
  - Collective operations are more efficient than p2p, and should be used if possible

Point-to-point Operations

- Intel MPI PingPong benchmark: BG/L co-mode vs. BG/P smp-mode
- Nearest neighbor communication
- The break line is due to switching from short to eager



IBM System Blue Gene Solution: Blue Gene/P Application Development RedBook

BlueGene/P Collective Operations

- Intel MPI Collective Benchmark
- Preferred over P2P due to lower overhead, independent on mapping

MPI Routine	Condition	Network	Performance
MPI_Barrier	MPI_COMM_WORLD	barrier (global interrupt) network	1.2 µs
MPI_Barrier	any communicator	torus network	30 µs
MPI_Broadcast	MPI_COMM_WORLD	collective network	817 MB/sec
MPI_Broadcast	rectangular communicator	torus network	934 MB/sec
MPI_Allreduce	MPI_COMM_WORLD fixed-point	collective network	778 MB/sec
MPI_Allreduce	MPI_COMM_WORLD floating point	collective network	98 MB/sec
MPI_Alltoall[v]	any communicator	torus network	84-97% peak
MPI_Allgatherv		torus network	same as broadcast

IBM System Blue Gene Solution: Blue Gene/P Application Development RedBook

Personality of BlueGene/P*

```
#include <common/bgp_personality.h>
#include <common/bgp_personality_inlines.h>

_BGP_Personality_t p;

Kernel_GetPersonality( &p, sizeof(p) );

p.DDR_Config.DDRSizeMB;             /* memory size */
p.Kernel_Config.ProcessConfig;  /* running mode */
p.Network_Config.Xnodes;            /* torus dimensions */
p.Network_Config.Ynodes;
p.Network_Config.Znodes;
```

mpixlc_r -I/bgsys/drivers/ppcfloor/arch/include …

Overview

- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

Performance Toolkits

- Compiler options for profile information
  - no instrumentation, simple to use
  - -pg
  - gprof <exe> gmon.out.0
- TAU - Tuning and Analysis Utilities
  - /soft/apps/tau/tau-latest
  - requires additional instrumentation
  - extensive visualization capabilities
  - can be combined with PAPI-3.9.0 hardware counters*
- HPCT - IBM High-Performance Computing Toolkit
  - /soft/apps/hpct_bgp
  - New product, not much feedback is available, esp. for large projects
  - MPI profiling and tracing tool, CPU Profiling, Hardware Counter Performance Monitoring, I/O Performance

Use of gprof Tool with Compiler Options

- Profiling is collecting and arranging statistics of running program
- Simple to use: does NOT require instrumentation of sources
- Use -p option at compile AND link time
- Use -g option, but remember that it removes automatic inlining
- Run program: it will produce gmon.out.N binary files, one for each MPI task
- Convert a binary to readable text format:

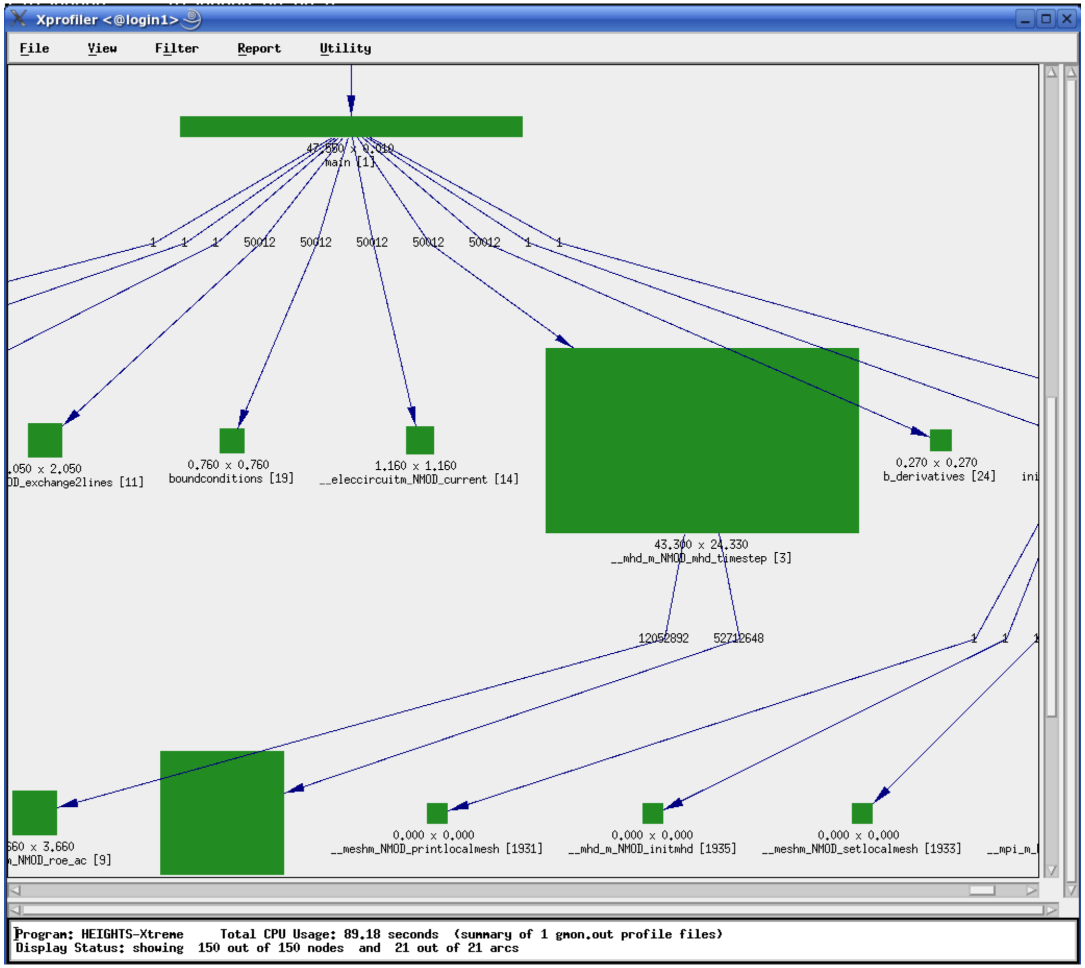
  gprof <executable> gmon.out.0
- Alternatively, use Xprofiler graphical tool (part of HPCT)
- http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html

Flat profile

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  s/call   s/call  name
 32.49     24.33    24.33    50012     0.00     0.00  __mhd_m_NMOD_mhd_timestep
 20.45     39.64    15.31 52712648     0.00     0.00  __thermom_NMOD_roe_peta
  7.09     44.95     5.31                             DCMF::hwBarrier::poll()
  7.08     50.25     5.30                             DMA_RecFifoSimplePollNormalFifoById
  4.89     53.91     3.66 12052892     0.00     0.00  __thermom_NMOD_roe_ac
  3.02     56.18     2.27
   DCMF::Queueing::Lockbox::LockboxMessage::advance()
  2.74     58.23     2.05    50012     0.00     0.00  __mpi_m_NMOD_exchange2lines
  2.27     59.93     1.70
   DCMF::Protocol::MultiSend::TreeAllreduceRecvPostMessage::advanceDeep(DCMF::Queueing::Tree::TreeM
   sgContext)
  1.80     61.28     1.35                             DCMF::DMA::Device::advance()
  1.55     62.44     1.16    50012     0.00     0.00  __eleccircuitm_NMOD_current
  1.32     63.42     0.99                             DCMF::Queueing::Lockbox::Device::advance()
  1.23     64.34     0.92                             DCMF_Messager_advance
  0.05     73.54     0.04                             DCMF_Send
  0.05     73.58     0.04                             MPIDI_BG2S_RecvCB
  0.05     73.62     0.04                             DCMF::DMA::Device::processAdvanceQueue()
```

- Search for functions with larger time usage
- Search for functions with larger number of calls

HPCT GUI Tool - Xprofiler



*TAU toolkit**

- Tuning and Analysis Utilities (TAU): A toolkit for performance evaluation, such as profiling, and tracing, and analysis of parallel programs
- Profiling
  - summary statistics of performance metrics
  - performance behavior of functions, blocks, calls
  - identifies bottlenecks and hot spots
  - implemented through sampling and/or instrumentation
- Tracing
  - when and where significant points (events) took place
  - saves information about each events
  - used to reconstruct dynamics of the program
  - requires code instrumentation

* Performance Research Lab, University of Oregon

Steps of TAU Performance Evaluation

- Collecting basic routine-level timing profile to determine where most time is being spent
- Collecting routine-level hardware counter data to determine types of performance problems
- Collecting callpath profiles to determine sequence of events causing performance problems
- Conducting fine-grained profiling and tracing to pinpoint performance bottlenecks (hardware counters, communications,…)

Using TAU: Basic steps

- Instrument the source code

  TAU includes tau_XXX scripts for automatic instrumentation:

  % mpicxx -o computePi computePi.cpp

  changed to

  % export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp/lib/
  Makefile.tau-multiplecounters-mpi-papi-pdt

  % tau_cxx.sh -o computePi computePi.cpp
- Execute MPI program as usual
- Obtain profile.NNN files, one for each MPI task
- Post-process profiles by console-based pprof utility
- Post-process profiles with GUI paraprof utility

Using TAU: Manual Instrumentation

- Initialization and runtime configuration

    TAU_PROFILE_INIT, TAU_PROFILE_SET_NODE

- Register a function to profile

    TAU_PROFILE

- Start/stop profiling

    TAU_START, TAU_STOP

- User-defined timing

  TAU_PROFILE_TIMER, TAU_PROFILE_START, TAU_PROFILE_STOP

- User-defined events

    TAU_REGISTER_EVENT, TAU_PROFILE_STMT

- Heap memory tracking

    TAU_TRACK_MEMORY, TAU_SET_INTERRUPT_INTERVAL

Specific of TAU on BG/P

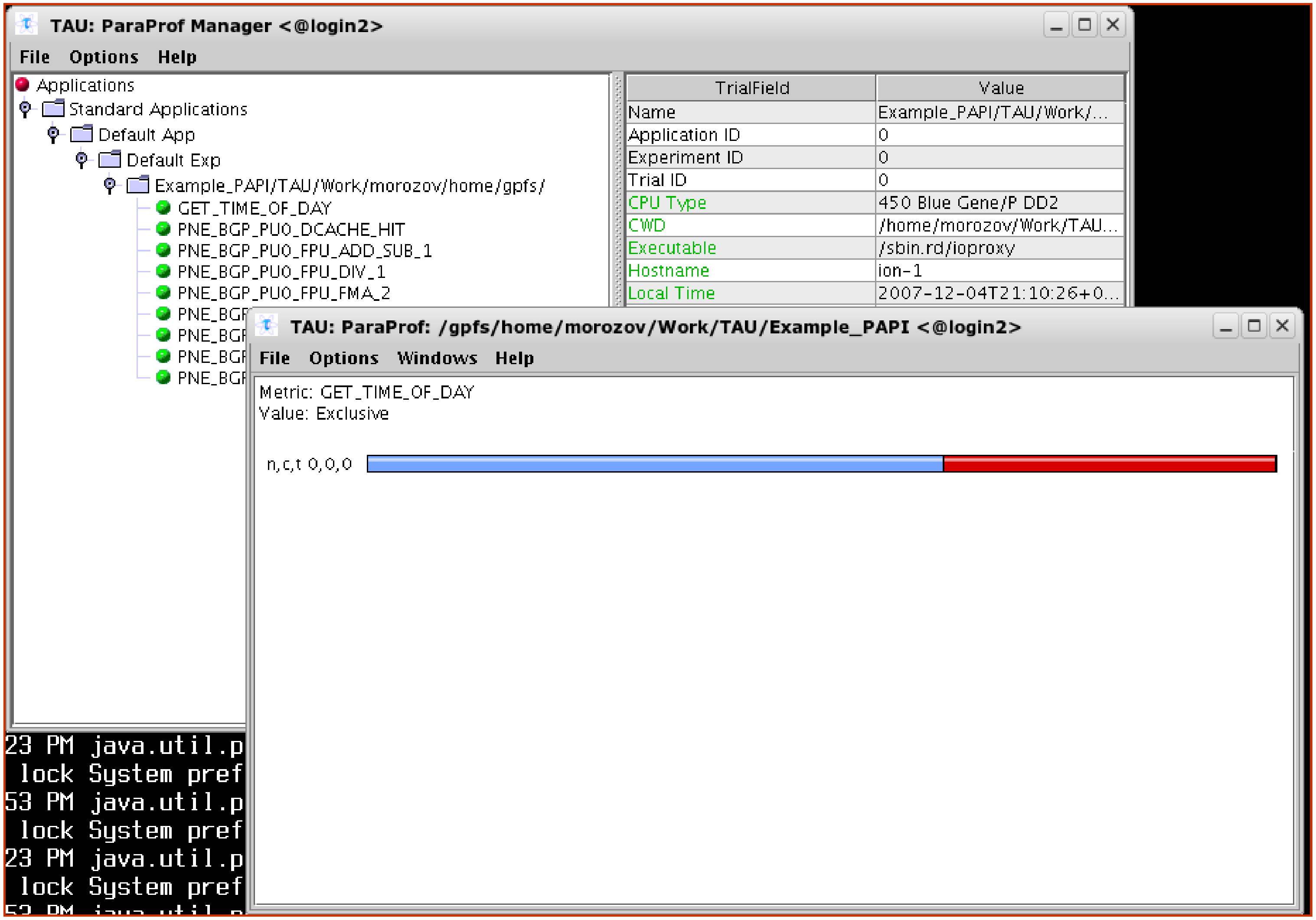
- Front end nodes are ppc64
- Back end nodes are bgp
- TAU interactive tools are built for ppc64 or Java
- Back end tools (measurement) are built for bgp
- Available configurations
  - TAU with PDT - profiling and tracing of functions
  - MPI - profiling and tracing only communication routines
  - pthreads - profiling threads
  - callpath - constructing functions call path
  - hardware counters - including PAPI-based counters support

pprof output

FUNCTION SUMMARY (mean):

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	62	16,643	1	2318.47	16643178	MAIN
93.2	15,508	15,508	62.5	0	248128	MULTIPLY_MATRICES
3.1	507	507	125.969	0	4032	MPI_Recv()
2.3	376	376	2000	0	188	MPI_Bcast()
0.7	116	116	1	0	116762	MPI_Finalize()
0.3	50	50	1	0	50779	MPI_Init()
0.1	11	11	125.969	0	91	MPI_Send()
0.1	9	9	0.03125	0	301331	INITIALIZE
0.0	0.00803	0.00803	1	0	8	MPI_Comm_rank()
0.0	0.006	0.006	1	0	6	MPI_Comm_size()

TAU GUI Tool - Paraprof



IBM HPCT Tool for MPI/CPU/IO Profile

- IBM High Performance Computing Toolkit - HPCT
  - Tools to visualize and analyze your performance data
  - Xprofiler and HPCT GUI instructions
  - Tools to optimize your application's performance

- MPI Performance: MPI Profiling and Tracing (mpitrace)
- CPU Performance: -pg and gmon.out.X, XProfiler, HPM
- Hardware Counter Performance Monitoring: HPM
- I/O Performance: I/O Profiling
- Threading Performance: OpenMP profiling
- Visualization and analysis: PeekPerf

HPCT: Message Passing Performance

- Implemented as PMPI wrappers around MPI functions
- No changes in source code
- Compile with -g, link with libmpitrace.a
- Captures MPI calls with source code traceback
- Does not synchronize MPI calls
- Generate XML output with PeekPerf

HPCT: Message Passing Performance

MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Comm_size	1 (1)	0 ... 4	0	1E-07
MPI_Comm_rank	1 (1)	0 ... 4	0	1E-07
MPI_Isend	2 (1)	0 ... 4	3	0.000006
MPI_Isend	2 (2)	5 ... 16	12	1.4E-06
MPI_Isend	2 (3)	17 ... 64	48	1.3E-06
MPI_Isend	2 (4)	65 ... 256	192	1.3E-06
MPI_Isend	2 (5)	257 ... 1K	768	1.3E-06
MPI_Isend	2 (6)	1K ... 4K	3072	1.3E-06
MPI_Isend	2 (7)	4K ... 16K	12288	1.3E-06
MPI_Isend	2 (8)	16K ... 64K	49152	1.3E-06
MPI_Isend	2 (9)	64K ... 256K	196608	1.7E-06
MPI_Isend	2 (A)	256K ... 1M	786432	1.7E-06
MPI_Isend	1 (B)	1M ... 4M	1048576	9E-07

MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Irecv	2 (1)	0 ... 4	3	4.7E-06
MPI_Irecv	2 (2)	5 ... 16	12	1.4E-06
MPI_Irecv	2 (3)	17 ... 64	48	1.5E-06
MPI_Irecv	2 (4)	65 ... 256	192	2.4E-06
MPI_Irecv	2 (5)	257 ... 1K	768	2.6E-06
MPI_Irecv	2 (6)	1K ... 4K	3072	3.4E-06
MPI_Irecv	2 (7)	4K ... 16K	12288	7.1E-06
MPI_Irecv	2 (8)	16K ... 64K	49152	2.23E-05
MPI_Irecv	2 (9)	64K ... 256K	196608	9.98E-05
MPI_Irecv	2 (A)	256K ... 1M	786432	0.00039
MPI_Irecv	1 (B)	1M ... 4M	1048576	0.000517
MPI_Waitall	21 (1)	0 ... 4	0	1.98E-05
MPI_Barrier	5 (1)	0 ... 4	0	7.8E-06

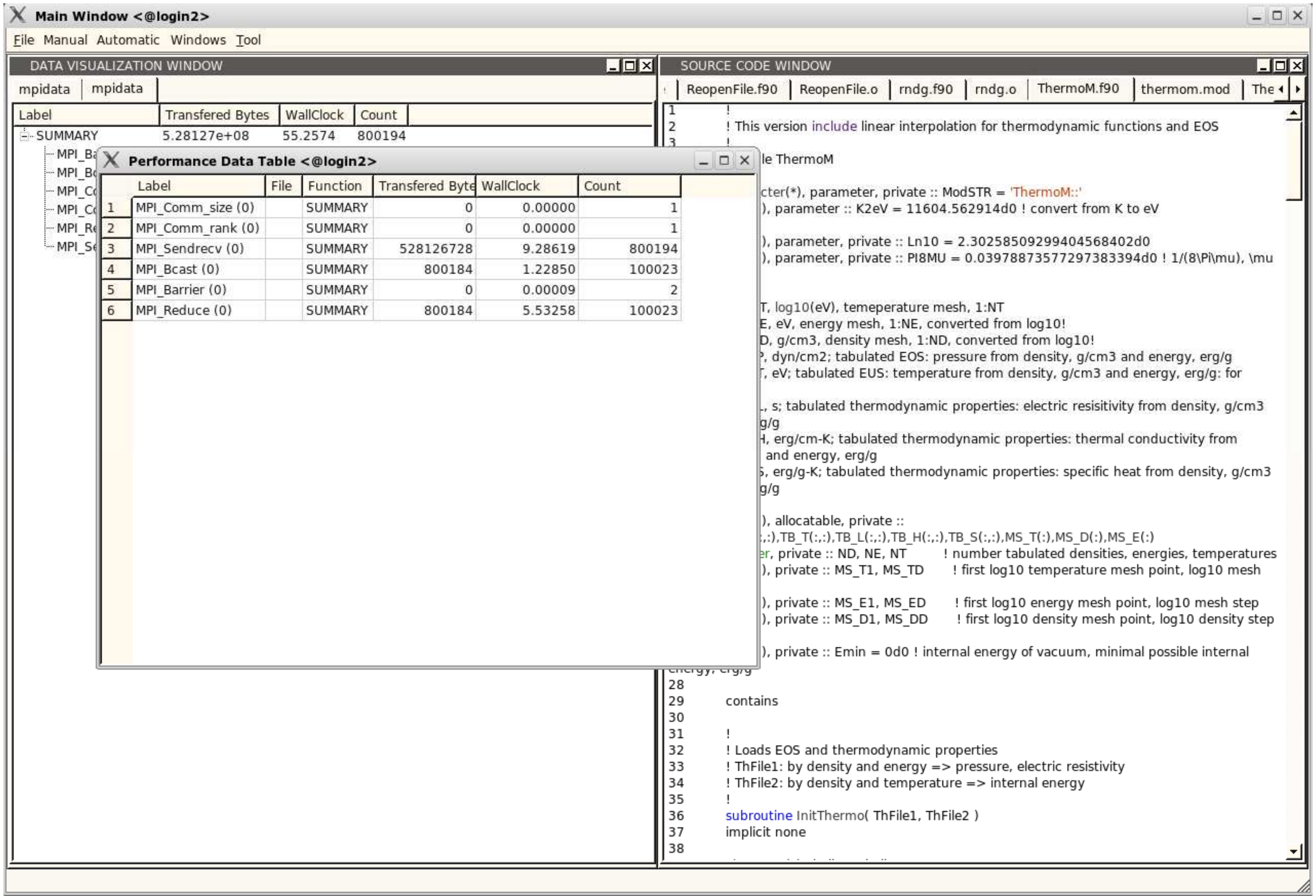
Example of using HPCT Tool

- Instrument the program
- See listing
- Use hardware counters
- Change optimization options

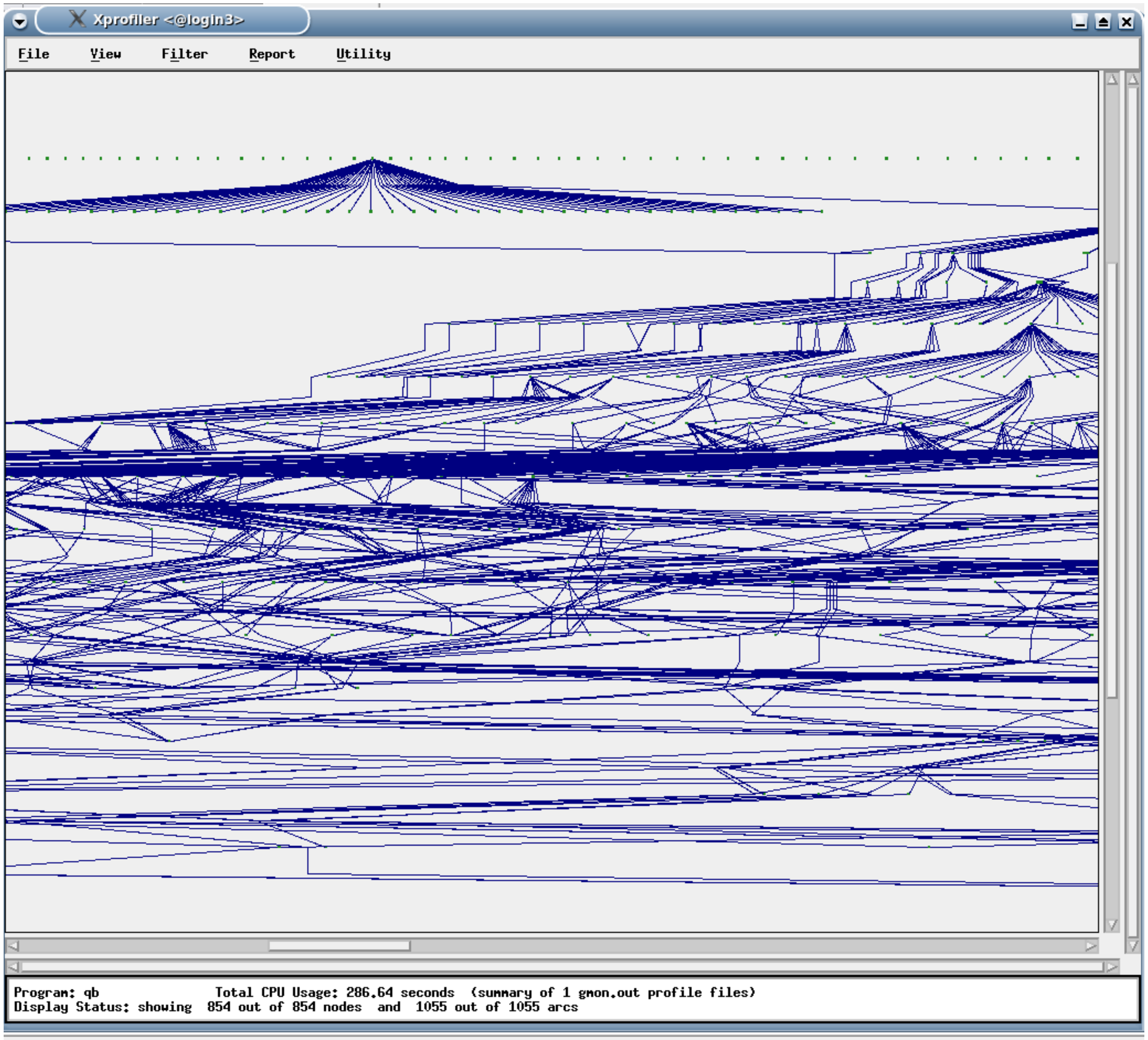
```
#include "libhpm.h"

hpmInit( taskID, "hpct-hcpm" );
hpmStart( 1, "multiply-regular" );
  for (i = 0; i < SIZE; i ++)
      for (j = 0; j < SIZE; j++)
          for (k = 0; k < SIZE; k++)
              C[i][j] += A[i][k] * B[k][j];
hpmStop( 1 );
hpmTerminate( taskID );
```

```
HPCT_DIR=/soft/apps/hpct_bgp
mpixlcxx_r -I$HPCT_DIR/include -o Hello Hello.cxx -L$HPCT_DIR/lib
-lhpm -llicense
```

HPCT GUI Tool - Peekperf



HPCT GUI Tool - XProfiler



Overview

- Application Developers' view
- Compiling and Building Tools
- I/O
- Scheduling and Running Jobs
- Optimization Techniques
- Performance Tools
- Debugging Tools

Debugging on BlueGene/P

- GDB and Totalview (Totalview technologies) are available
- isub launcher should be used:
  - isub -t 30 -n 64 -A myproject -q short
  - waiting the job to start … prompt

    gdb <mpi_args>

    totalview <tv_args> mpirun -a <mpi_args>

    quit

isub and mpirun arguments

- isub
  - Require: -q queue -A Project -t time
  - Optional: -K kernel
- mpirun
  - Require: -np CPU (not --proccount)
  - Require: -mode mode (not --mode)
  - Recommended: -verbose
  - Recommended: -nofree (experts only)
- example launch

gdb -np 64 -mode smp -verbose 2 program

>

gdb server

- [Partition boots]
- [wait for prompt]
- Type one of the following
  - **<rank>** to get a connection to that rank
  - **dump_proctable** to get all rank IP:PORT info
- [Start client in other windows]
- Hit <return> to start the program

gdb client

- [gdb server already started]
- [dump_proctable on server gives IP:PORT]
- Attach one client to each interesting task

  /bgsys/drivers/ppcfloor/gnu-linux/bin/gdb

  target remote <IP:PORT>

  [client waits for server to become active]
- [return to server and hit <enter>]
- [clients will give a prompt]

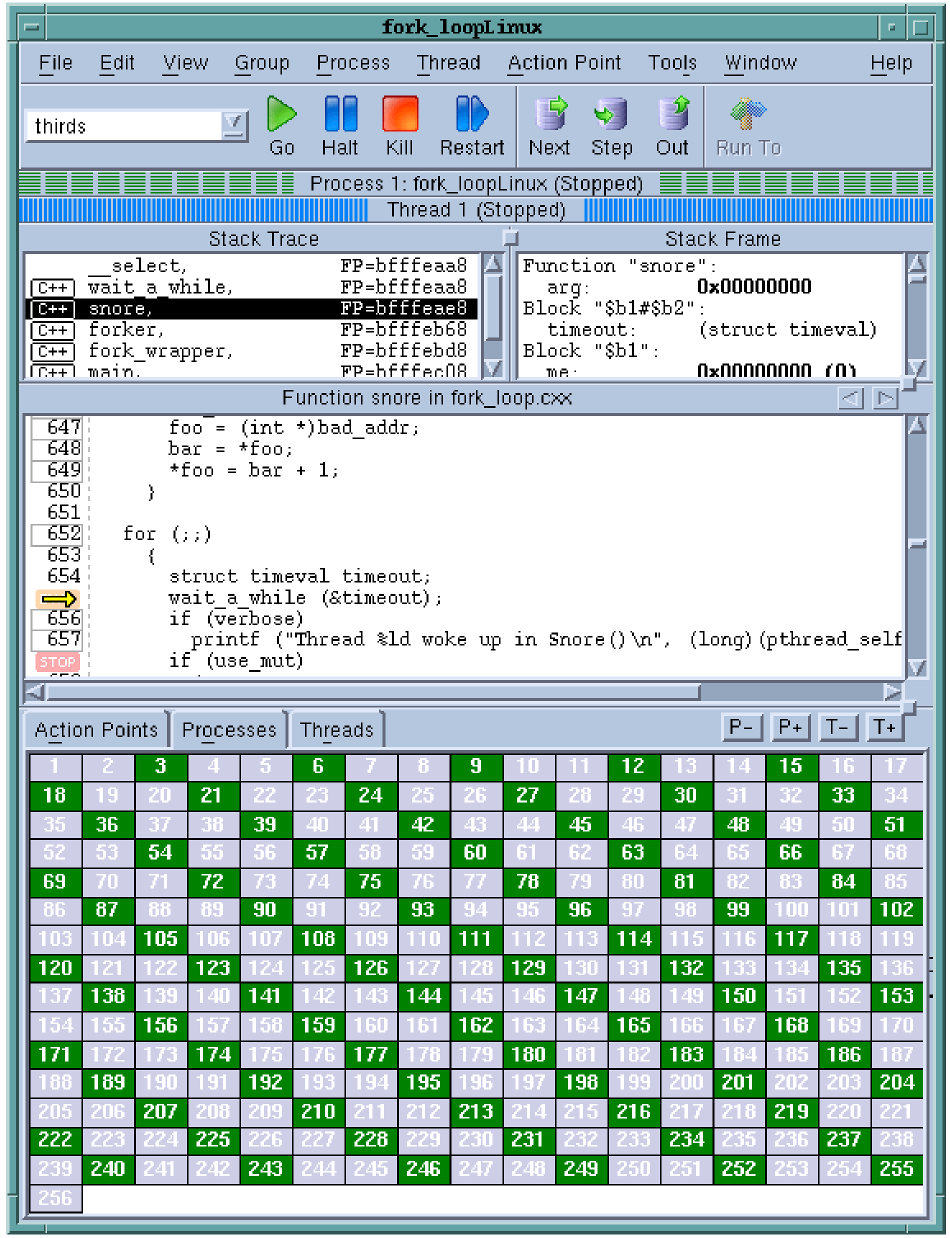
gdb commands

- break <nnn>
  - info break
  - delete
- next
- print
- where
- continue

more info: http://www.gnu.org/software/gdb/documentation/

Totalview

- C, C++, Fortran
- wide compiler/platform support
- multi-threaded debugging
- parallel debugging
- remote debugging
- memory debugging
- Extensive GUI
- CLI for scripting and batch



Starting Totalview

- Use ssh -X to login with X11 tunnel created
- Add +totalview to ~.softenvrc file
- To submit a job for debugging

  isub <qsub_args> --run totalview <tv_args> mpirun -a <mpi_args>

  e.g.

  isub -t 30 -n 64 -A Project --run totalview mpirun -a np 64 program
- Totalview will start when job is allocated
- Typically just hit "Go" button
- Wait while partition boot
- All processes are halted after execution single instruction
- Set necessary breakpoints and continue, inspect processes when needed

Specific of Totalview on BG/P

- Dynamic and multi-threaded applications are in beta stage
- Memory debugging is at initial stage
- BG/P message queue are unavailable
- Core files are unsupported
  - BG/P uses Lightweight Core File (LCF) format
  - Use bgp_stack, coreprocessor.pl

Tuning code for BlueGene/P

- Structuring data in adjacent pairs
  - Allows to use quadword load/store operations
- Using vectorizable blocks
  - Organize the code sequences with single entry point
  - Minimize branching for special cases (exceptions, NaN values)
  - Minimize dependencies between blocks
- Minimize the usage of C/C++ pointers, guarantee disjoint references
- Use inline (with caution)
  - to remove overhead with brunching
  - to enlarge the vectorizable blocks
- Turn off range checking -qfloat=norngchk (with caution)

Resources

- ALCF Resource page
  http://www.alcf.anl.gov/support/usingALCF/index.php

- Getting Started
  http://www.alcf.anl.gov/support/gettingstarted/index.php

- IBM RedBooks:
  Compiler User Guides, Application Development Manuals
  http://www.redbooks.ibm.com/redbooks.nsf/redbooks/